Car vs Bike Image Classification

WISDOM IZUCHUKWU ADIKE

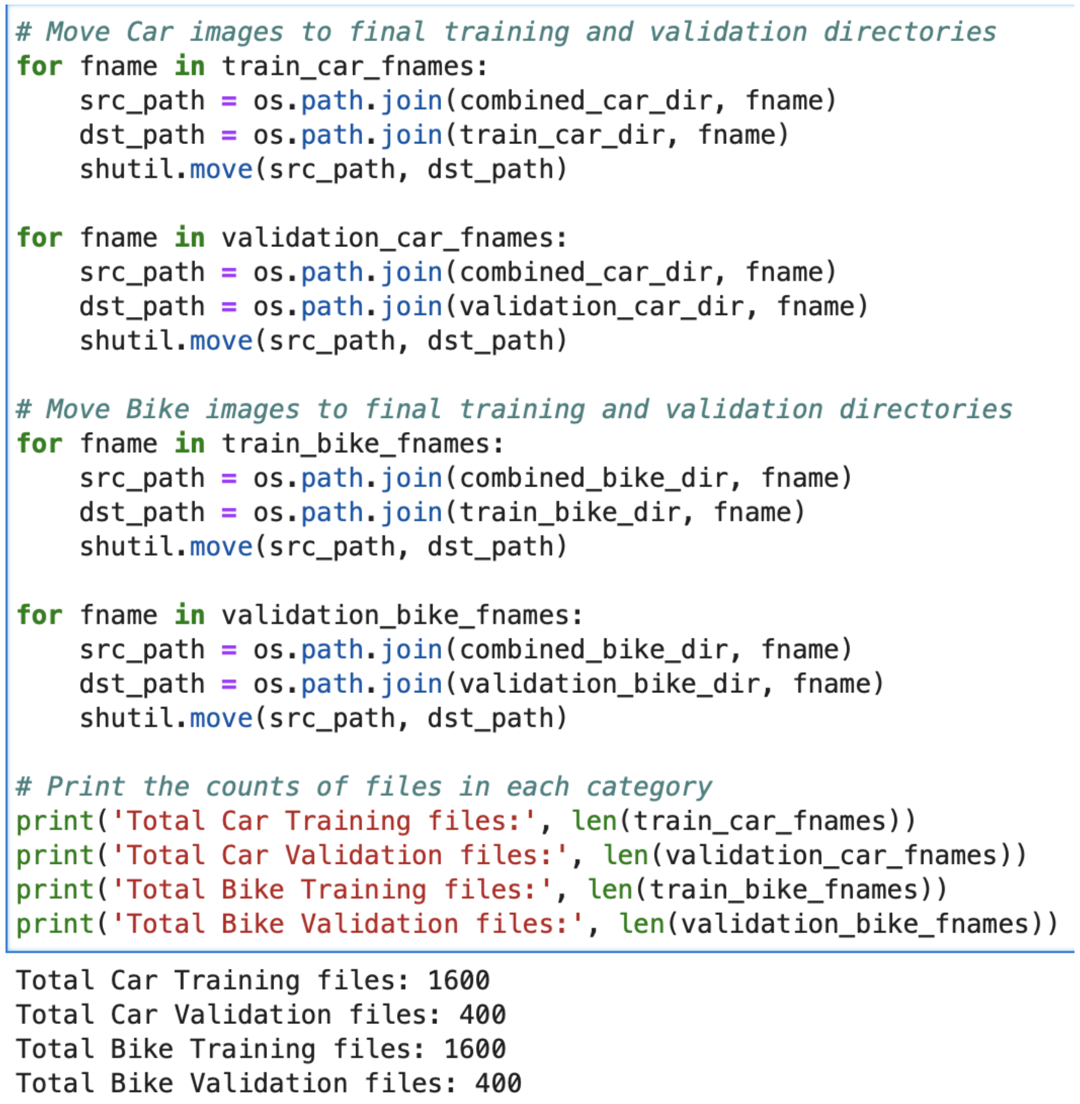
Contents

Car and Bike Image Description

Introduction

The dataset contains a total of 4000 images. The images have been labeled as either "Car" or "Bike" and are stored in separate directories. This dataset can be used for a variety of tasks related to image classification, including developing and testing deep learning algorithms, evaluating the effectiveness of different image features and classification techniques, and comparing the performance of different models. While collecting these images, it was made sure that all types of bikes and cars are included in the image collection. This is because of the high Intra-variety of cars and bikes. That is, there are different types of cars and bikes, which make it a little tough task for the models because the model will also have to understand the high variety of bikes and cars.

Image Description 01



```
# Move Car images to final training and validation directories
for fname in train_car_fnames:
    src_path = os.path.join(combined_car_dir, fname)
    dst_path = os.path.join(train_car_dir, fname)
    shutil.move(src_path, dst_path)

for fname in validation_car_fnames:
    src_path = os.path.join(combined_car_dir, fname)
    dst_path = os.path.join(validation_car_dir, fname)
    shutil.move(src_path, dst_path)

# Move Bike images to final training and validation directories
for fname in train_bike_fnames:
    src_path = os.path.join(combined_bike_dir, fname)
    dst_path = os.path.join(train_bike_dir, fname)
    shutil.move(src_path, dst_path)

for fname in validation_bike_fnames:
    src_path = os.path.join(combined_bike_dir, fname)
    dst_path = os.path.join(validation_bike_dir, fname)
    shutil.move(src_path, dst_path)

# Print the counts of files in each category
print('Total Car Training files:', len(train_car_fnames))
print('Total Car Validation files:', len(validation_car_fnames))
print('Total Bike Training files:', len(train_bike_fnames))
print('Total Bike Validation files:', len(validation_bike_fnames))
```

```
Total Car Training files: 1600
Total Car Validation files: 400
Total Bike Training files: 1600
Total Bike Validation files: 400
```

Separating Images into training and validating directories.

Here, we successfully moved the images of cars and bikes from a combined directory into separate training and validation directories for each category (cars and bikes). After moving the images, it prints the counts of files in each category for both training and validation sets where we have a total of 1600 car training files, 400 car validation files, 1600 bike training files and 400 bike validation files.

Image Description 02

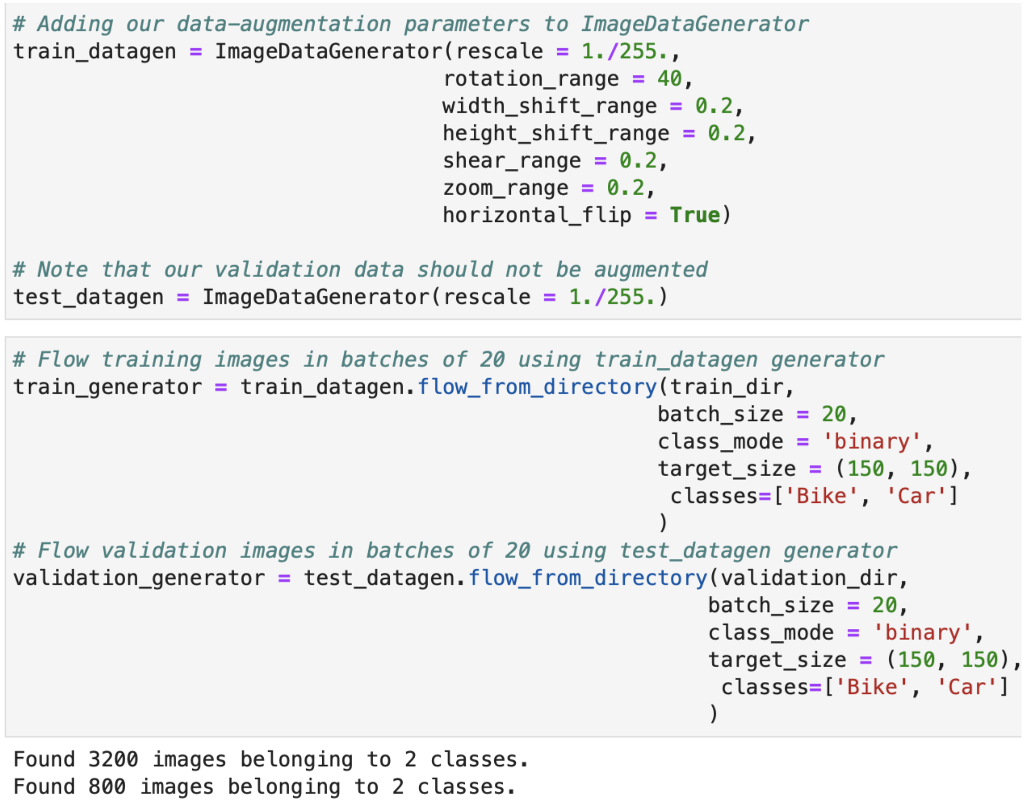
Samples of the some of the images of cars and bikes.

Main Objective of the analysis

In this report, I will be using deep learning models in classifying 2 distinct images 'Cars' and 'Bikes' gotten locally in my computer. Those images were separated into training and validation directories initially defined, where I performed data augmentation with image data generators. I used CNN based model for training the images and also pre-trained models such as InceptionV3 and VGG16. In the process of training, we set the layers of the pre-trained model as non-trainable (frozen) to retain their learned features while adding new layers for fine-tuning the model for binary classification.

Data-augmentation and Image Data Generators

Data-augmentation and Image Data Generators



```
# Adding our data-augmentation parameters to ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255.,
                                   rotation_range = 40,
                                   width_shift_range = 0.2,
                                   height_shift_range = 0.2,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

# Note that our validation data should not be augmented
test_datagen = ImageDataGenerator(rescale = 1./255.)
```

```
# Flow training images in batches of 20 using train_datagen generator
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size = 20,
                                                    class_mode = 'binary',
                                                    target_size = (150, 150),
                                                     classes=['Bike', 'Car']
                                                    )
# Flow validation images in batches of 20 using test_datagen generator
validation_generator = test_datagen.flow_from_directory(validation_dir,
                                                        batch_size = 20,
                                                        class_mode = 'binary',
                                                        target_size = (150, 150),
                                                         classes=['Bike', 'Car']
                                                        )
```

```
Found 3200 images belonging to 2 classes.
Found 800 images belonging to 2 classes.
```

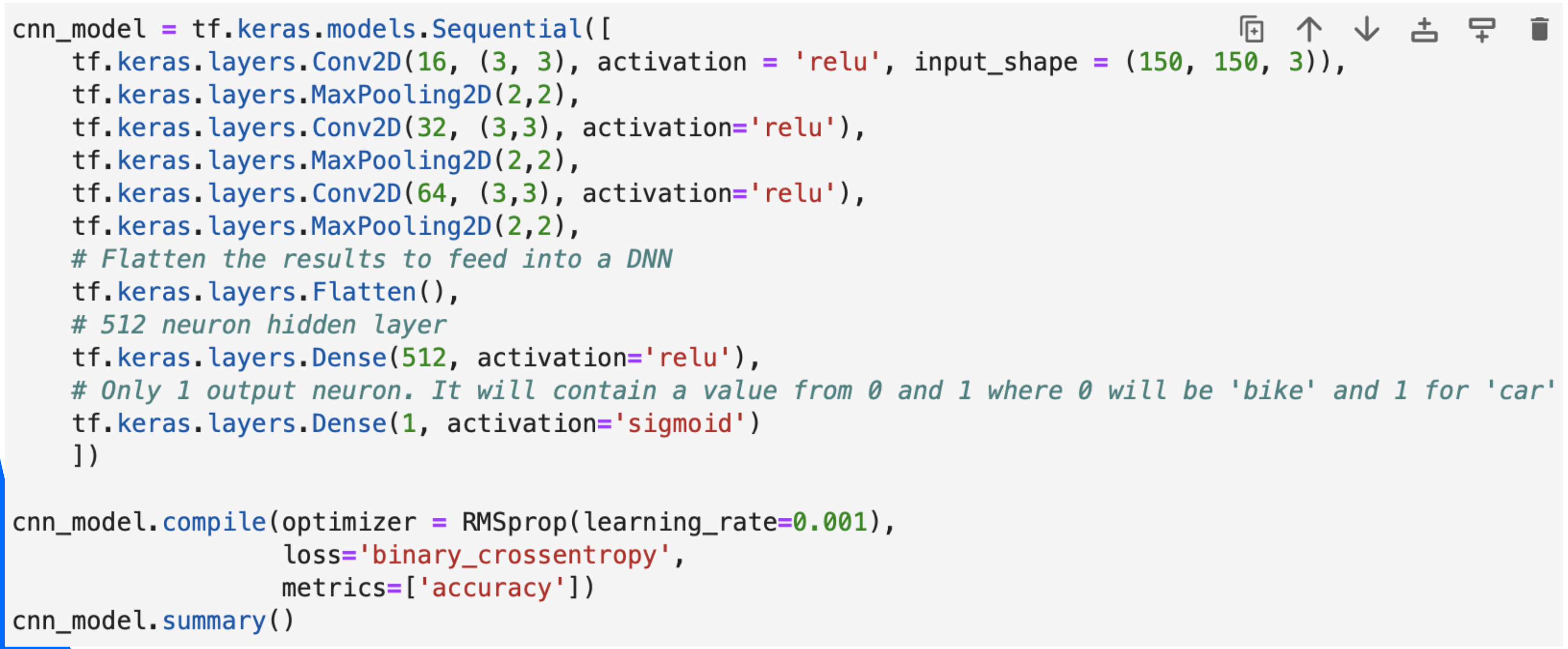
Performing data augmentation

Here, we set up data augmentation parameters for training images and created two data generators: one for training data with augmentation and another for validation data without augmentation. The target image size is 150x150 pixels, and the classes are defined as 'Bike' and 'Car'. Well, it discovered 3200 images belonging to 2 classes and 800 images also belonging to 2 classes.

Applying Deep Learning Models

Applying Deep learning Model 01(CNN-based Model)

CNN Based Model

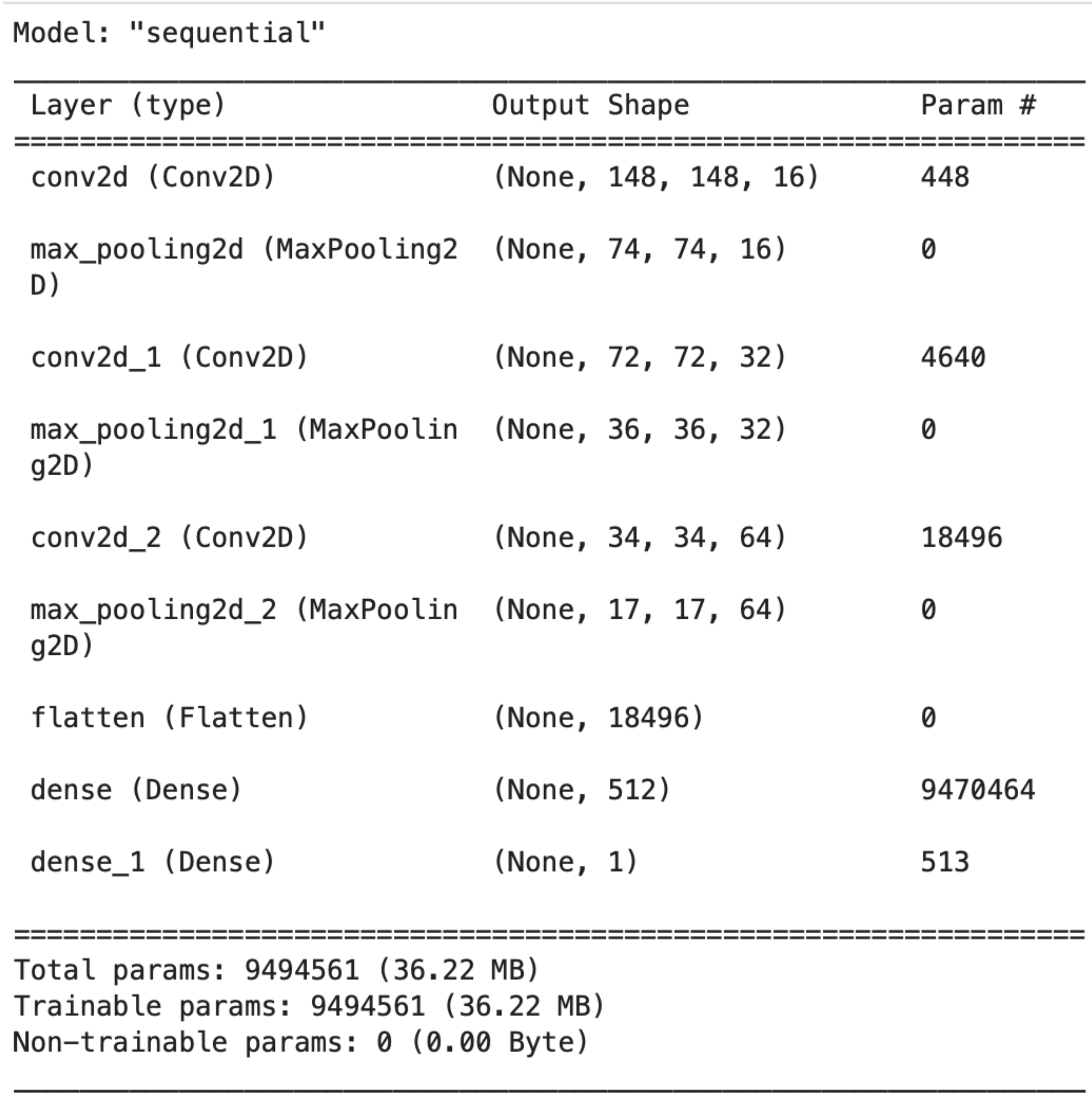


```
cnn_model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3, 3), activation = 'relu', input_shape = (150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    # Only 1 output neuron. It will contain a value from 0 and 1 where 0 will be 'bike' and 1 for 'car'
    tf.keras.layers.Dense(1, activation='sigmoid')
    ])

cnn_model.compile(optimizer = RMSprop(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
cnn_model.summary()
```

- Added a convolutional layer with 16 filters, ReLU activation, and input shape of (150, 150, 3).
- Added a max-pooling layer.
- Convolutional layer with 32 filters and ReLU activation.
- Added a max-pooling layer.
- Convolutional layer with 64 filters and ReLU activation.
- Max-pooling layer
- Flattening layer to prepare for a Dense Neural Network (DNN).
- A DNN layer with 512 neurons and ReLU activation.
- A final output layer with 1 neuron and sigmoid activation for binary classification.

Applying Deep learning Models 02 (CNN)

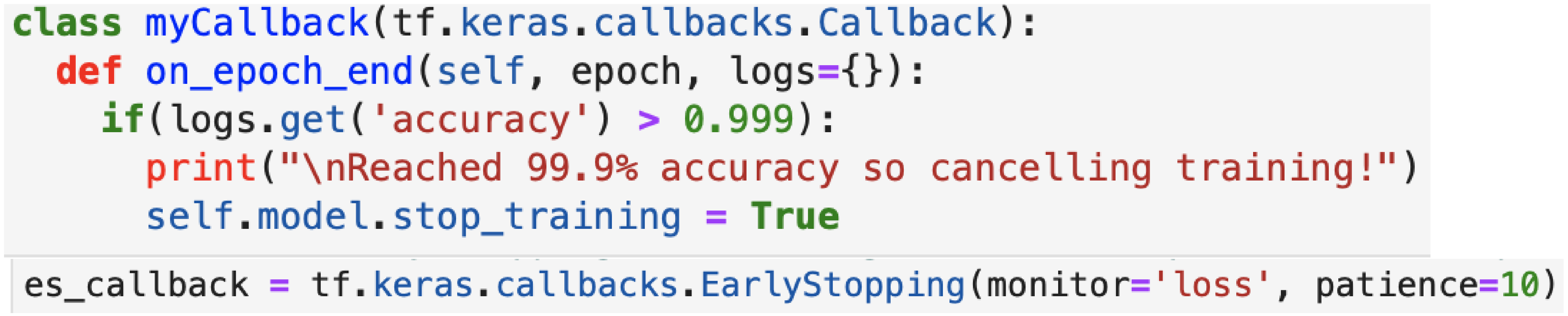
This model is a Sequential Convolutional Neural Network (CNN) architecture consisting of three convolutional layers with max-pooling, followed by a flattening layer and two dense layers. The total number of trainable parameters is 9,494,561 (approximately 36.22 MB), while there are no non-trainable parameters.



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 16)      448

 max_pooling2d (MaxPooling2  (None, 74, 74, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 72, 72, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 36, 36, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 17, 17, 64)        0
 g2D)

 flatten (Flatten)           (None, 18496)             0

 dense (Dense)               (None, 512)               9470464

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 9494561 (36.22 MB)
Trainable params: 9494561 (36.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Applying Deep learning Models 03 (CNN)

Defining the callback and early stopping function

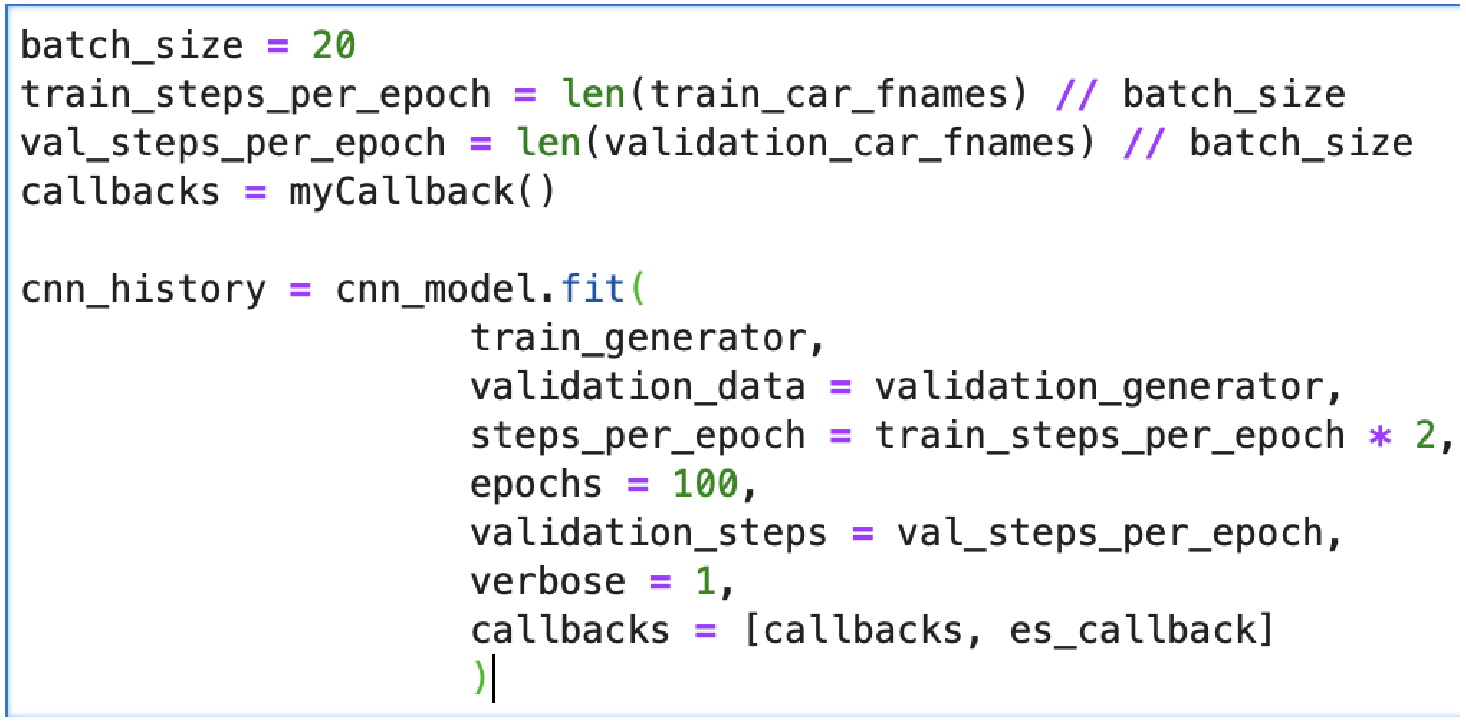


```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('accuracy') > 0.999):
      print("\nReached 99.9% accuracy so cancelling training!")
      self.model.stop_training = True
```

```
es_callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=10)
```

Defining a Callback class that stops training once accuracy reaches 99.9% so we don't want to overfit our model. While the Callback for early stopping, If training loss didn't improve after 10 epoch stop training

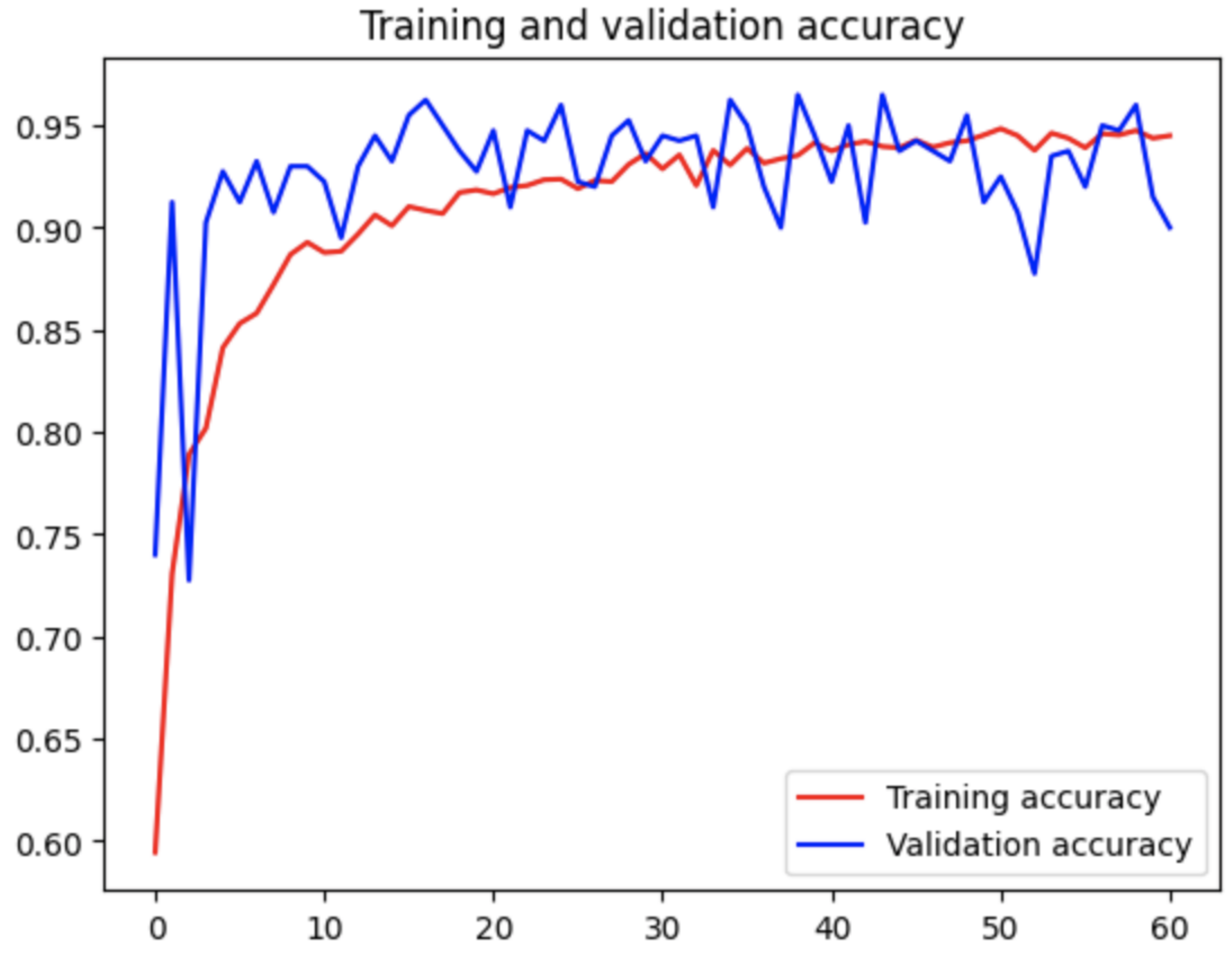
Applying Deep Learning Models 04 (CNN)



```
batch_size = 20
train_steps_per_epoch = len(train_car_fnames) // batch_size
val_steps_per_epoch = len(validation_car_fnames) // batch_size
callbacks = myCallback()

cnn_history = cnn_model.fit(
                    train_generator,
                    validation_data = validation_generator,
                    steps_per_epoch = train_steps_per_epoch * 2,
                    epochs = 100,
                    validation_steps = val_steps_per_epoch,
                    verbose = 1,
                    callbacks = [callbacks, es_callback]
                    )
```

Here, we sets the batch size to 20, calculates steps per epoch for training and validation, defines callbacks, and trains the CNN model using the specified parameters for 100 epochs with callbacks for monitoring and early stopping.

Applying Deep Learning Models 05 (CNN)

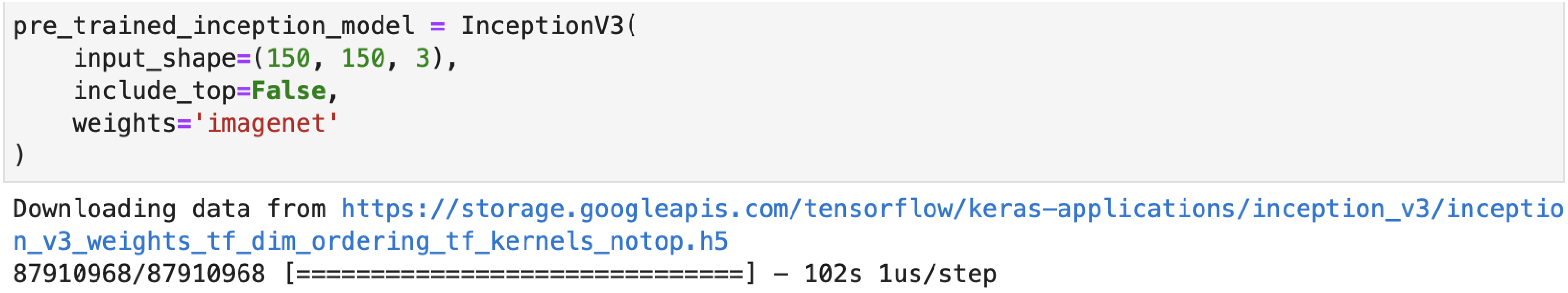


```
<Figure size 640x480 with 0 Axes>
```

Well, the training stopped after the 61st epoch with big improvement but we tried to still improve it by using Transfer learning.

Applying Deep Learning Models 06 (InceptionV3 Model)

InceptionV3 Model



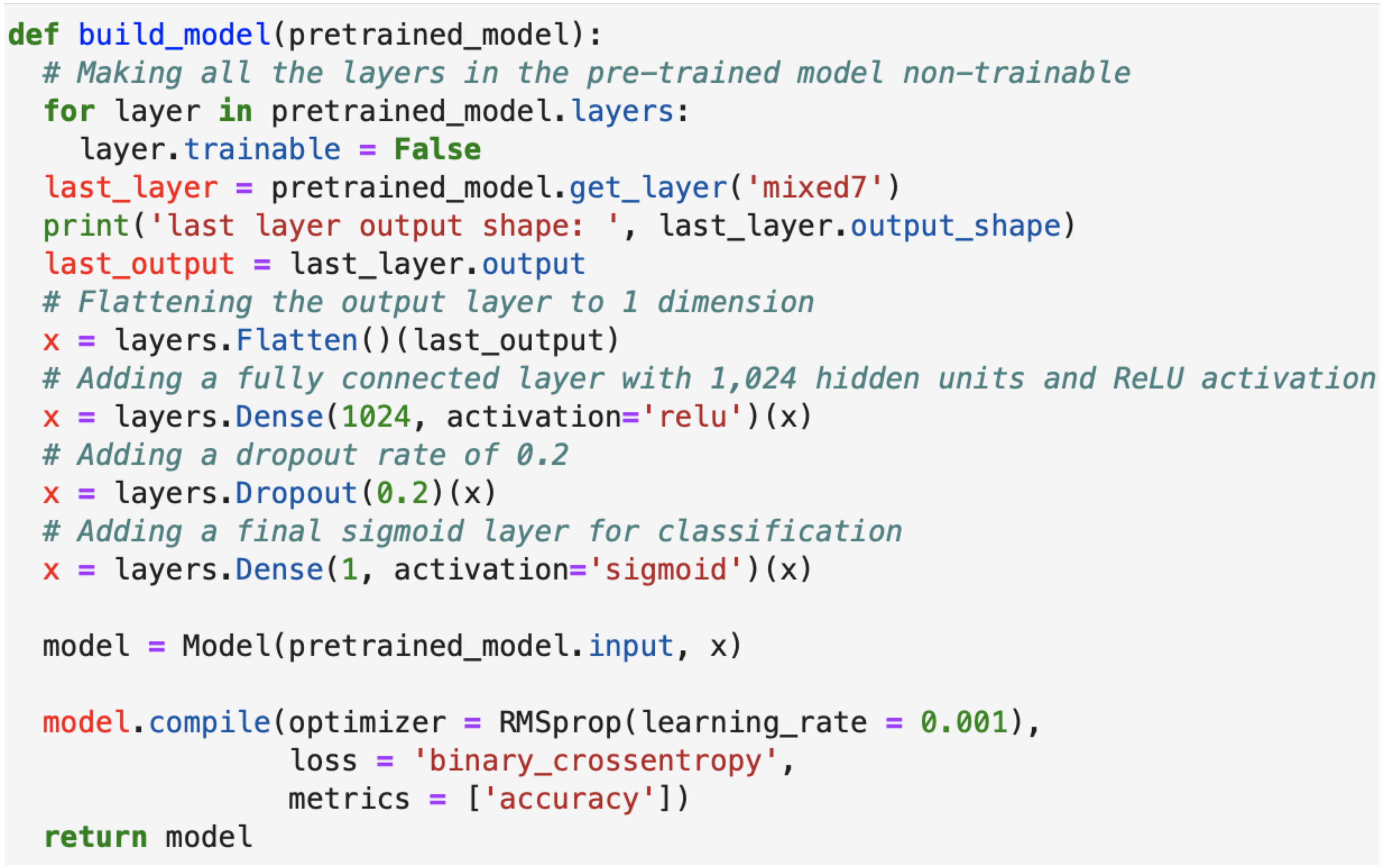
```
pre_trained_inception_model = InceptionV3(
    input_shape=(150, 150, 3),
    include_top=False,
    weights='imagenet'
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inceptio
n_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 [==============================] - 102s 1us/step
```

This code initializes a pre-trained InceptionV3 model with an input shape of 150x150 pixels and 3 color channels, excluding the top classification layer, and using pre-trained weights from the 'imagenet' dataset for feature extraction.

Applying Deep Learning Models 07 (InceptionV3)

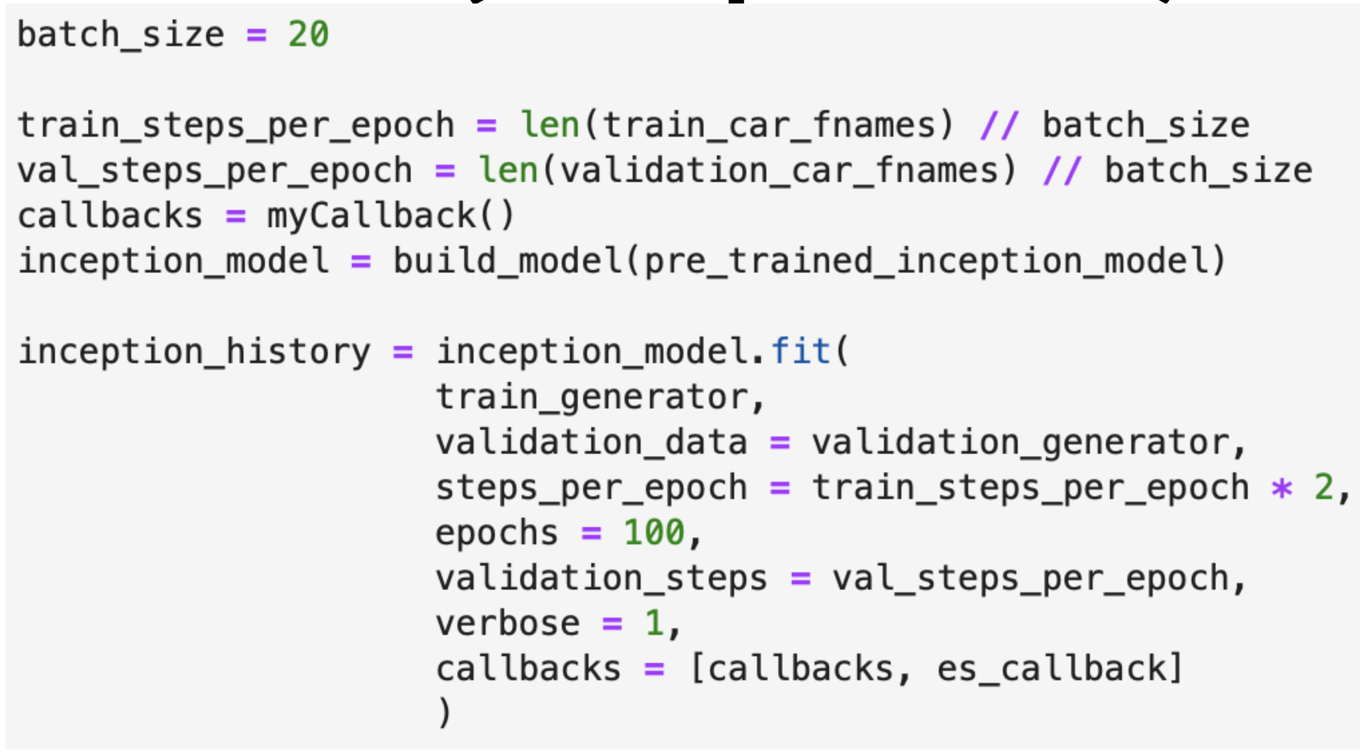
Functional API to merge both our model and the pretrained inception model



```python
def build_model(pretrained_model):
  # Making all the layers in the pre-trained model non-trainable
  for layer in pretrained_model.layers:
    layer.trainable = False
  last_layer = pretrained_model.get_layer('mixed7')
  print('last layer output shape: ', last_layer.output_shape)
  last_output = last_layer.output
  # Flattening the output layer to 1 dimension
  x = layers.Flatten()(last_output)
  # Adding a fully connected layer with 1,024 hidden units and ReLU activation
  x = layers.Dense(1024, activation='relu')(x)
  # Adding a dropout rate of 0.2
  x = layers.Dropout(0.2)(x)
  # Adding a final sigmoid layer for classification
  x = layers.Dense(1, activation='sigmoid')(x)

  model = Model(pretrained_model.input, x)

  model.compile(optimizer = RMSprop(learning_rate = 0.001),
                loss = 'binary_crossentropy',
                metrics = ['accuracy'])
  return model
```

This function builds a new neural network model based on a pre-trained model, setting all layers of the pre-trained model as non-trainable. It then adds additional layers for flattening, fully connected with 1,024 hidden units and ReLU activation, dropout with a rate of 0.2, and a final sigmoid output layer for binary classification. The model is compiled with RMSprop optimizer, binary cross-entropy loss, and accuracy metric, and it's returned as the final model.

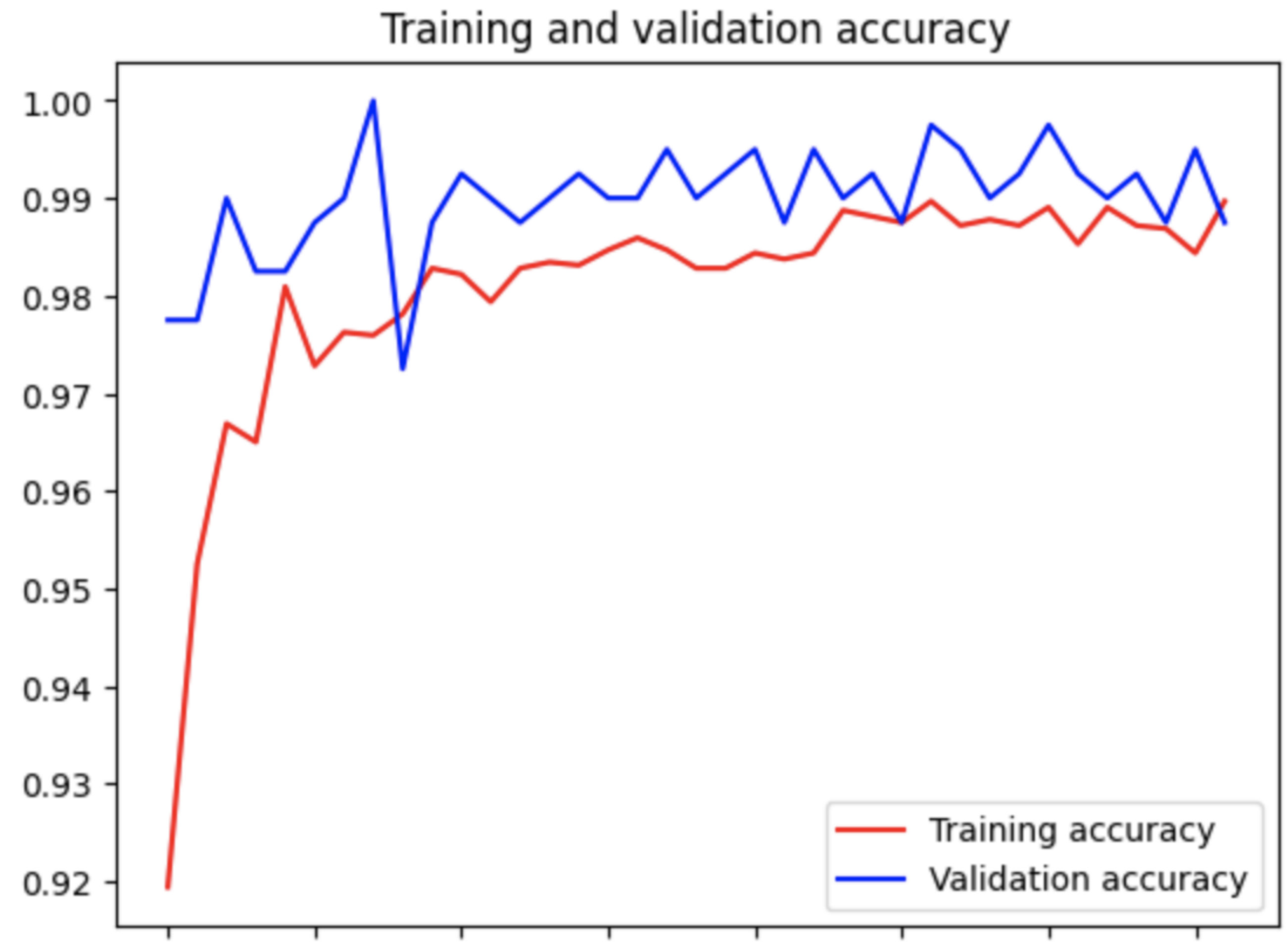
Applying Deep Learning Models 08 (InceptionV3)



```
batch_size = 20

train_steps_per_epoch = len(train_car_fnames) // batch_size
val_steps_per_epoch = len(validation_car_fnames) // batch_size
callbacks = myCallback()
inception_model = build_model(pre_trained_inception_model)

inception_history = inception_model.fit(
                    train_generator,
                    validation_data = validation_generator,
                    steps_per_epoch = train_steps_per_epoch * 2,
                    epochs = 100,
                    validation_steps = val_steps_per_epoch,
                    verbose = 1,
                    callbacks = [callbacks, es_callback]
                    )
```

Here, we sets the batch size to 20, calculates steps per epoch for training and validation, defines callbacks, and trains the pre-trained InceptionV3 model using the specified parameters for 100 epochs with callbacks for monitoring and early stopping.

Applying Deep Learning Models 09 (InceptionV3)

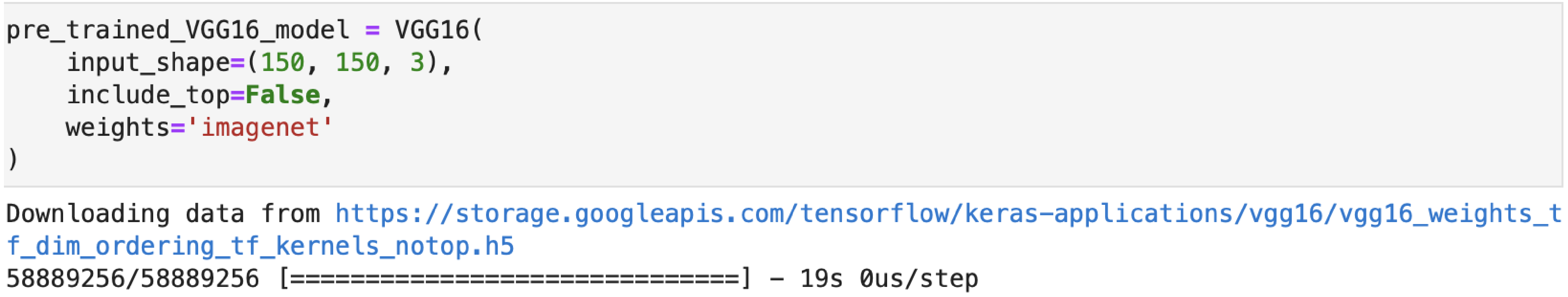


```
<Figure size 640x480 with 0 Axes>
```

Well, the training stopped after the 37th epoch with slight improvement.

Applying Deep Learning Models 10 (VGG16 Model)

VGG16 Model



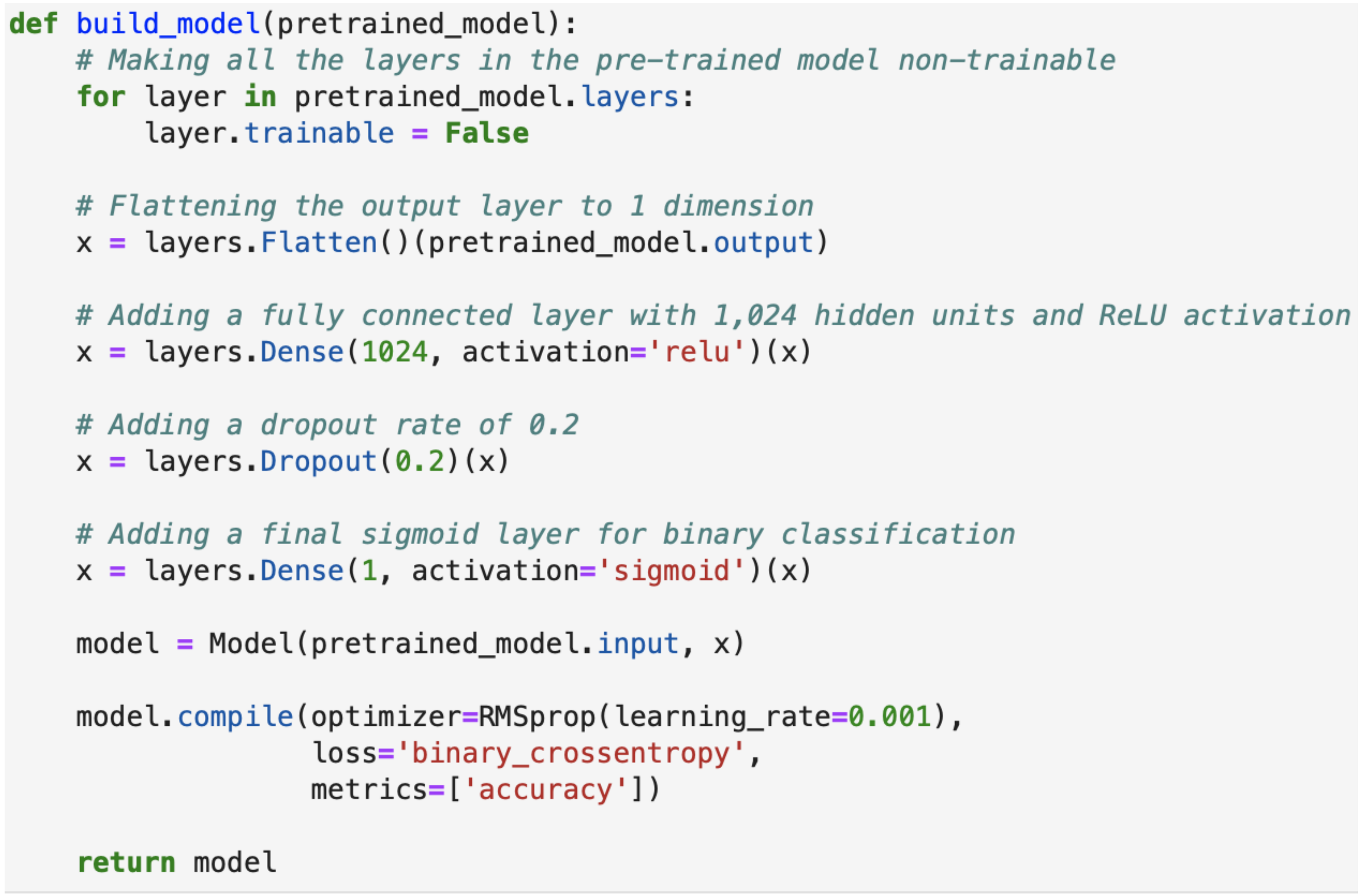
```
pre_trained_VGG16_model = VGG16(
    input_shape=(150, 150, 3),
    include_top=False,
    weights='imagenet'
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_t
f_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 19s 0us/step
```

This code initializes a pre-trained VGG16 model with an input shape of 150x150 pixels and 3 color channels, excluding the top classification layer, and using pre-trained weights from the 'imagenet' dataset for feature extraction.

Applying Deep Learning Models 11 (VGG16)

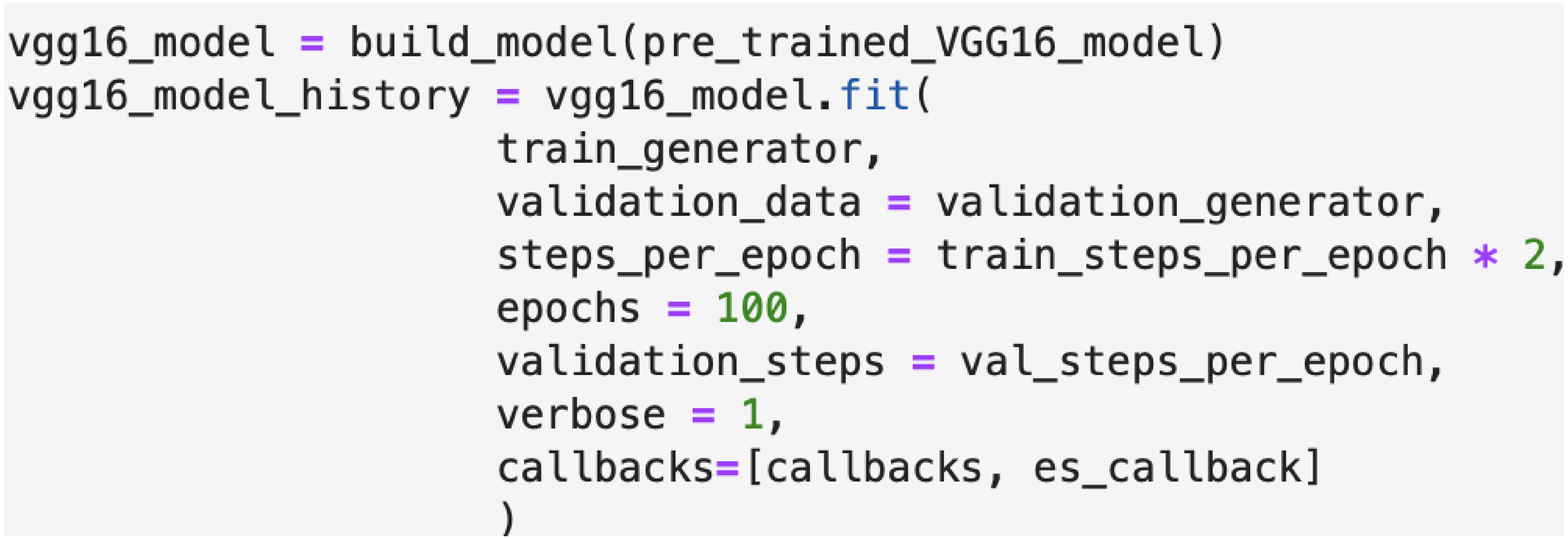
Functional API to merge both our model and the pretrained VGG16 model



```python
def build_model(pretrained_model):
    # Making all the layers in the pre-trained model non-trainable
    for layer in pretrained_model.layers:
        layer.trainable = False

    # Flattening the output layer to 1 dimension
    x = layers.Flatten()(pretrained_model.output)

    # Adding a fully connected layer with 1,024 hidden units and ReLU activation
    x = layers.Dense(1024, activation='relu')(x)

    # Adding a dropout rate of 0.2
    x = layers.Dropout(0.2)(x)

    # Adding a final sigmoid layer for binary classification
    x = layers.Dense(1, activation='sigmoid')(x)

    model = Model(pretrained_model.input, x)

    model.compile(optimizer=RMSprop(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

    return model
```

This function builds a new neural network model based on a pre-trained model, setting all layers of the pre-trained model as non-trainable. It then adds additional layers for flattening, fully connected with 1,024 hidden units and ReLU activation, dropout with a rate of 0.2, and a final sigmoid output layer for binary classification. The model is compiled with RMSprop optimizer, binary cross-entropy loss, and accuracy metric, and it's returned as the final model.

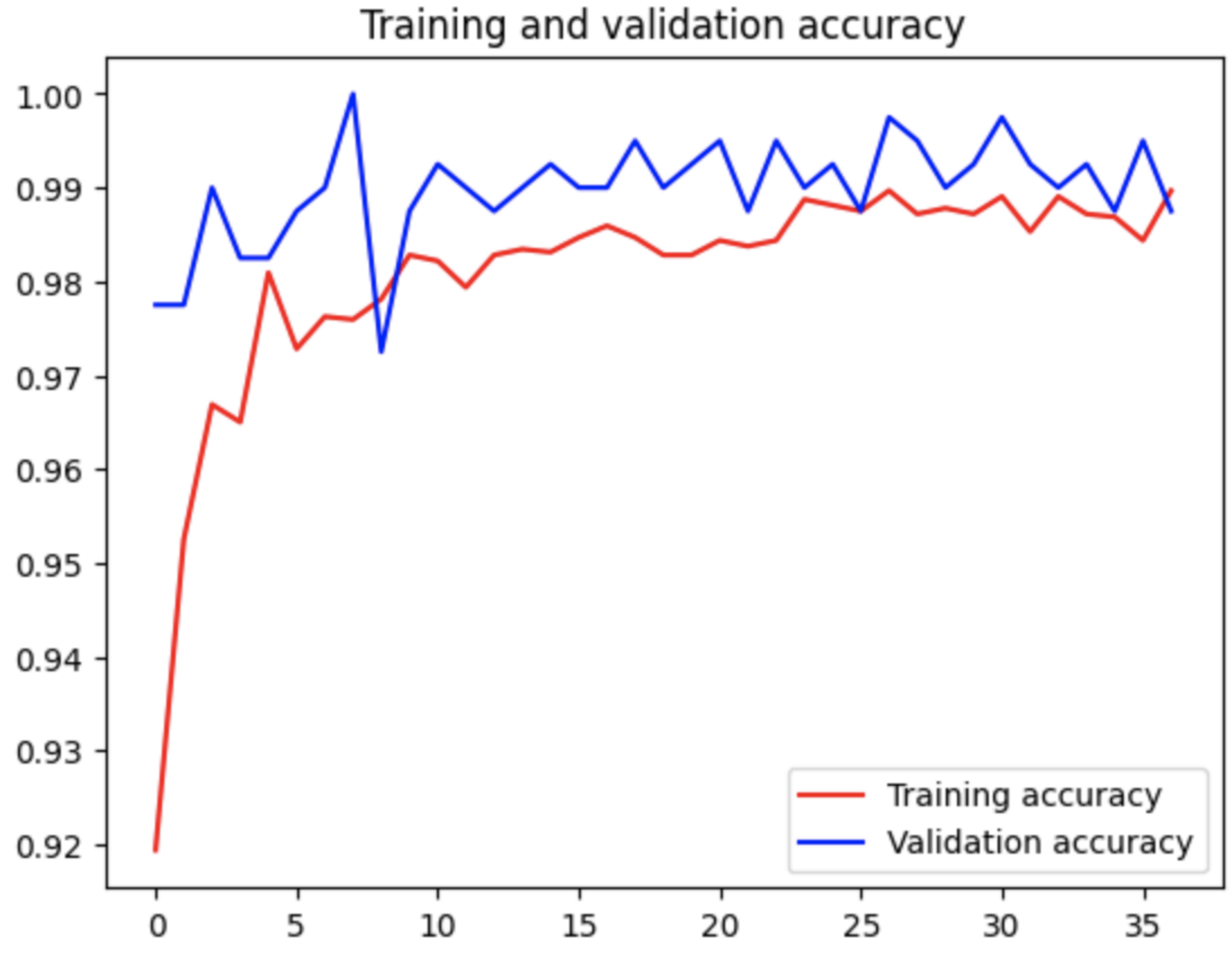
Applying Deep Learning Models 12 (VGG16)



```
vgg16_model = build_model(pre_trained_VGG16_model)
vgg16_model_history = vgg16_model.fit(
                    train_generator,
                    validation_data = validation_generator,
                    steps_per_epoch = train_steps_per_epoch * 2,
                    epochs = 100,
                    validation_steps = val_steps_per_epoch,
                    verbose = 1,
                    callbacks=[callbacks, es_callback]
                    )
```

Here, we also sets the batch size to 20, calculates steps per epoch for training and validation, defines callbacks, and trains the pre-trained VGG16 model using the specified parameters for 100 epochs with callbacks for monitoring and early stopping.

Applying Deep Learning Models 13 (VGG16)

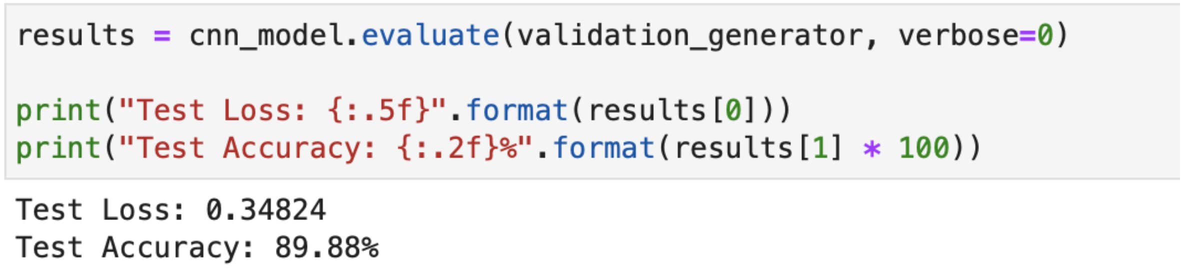


```
<Figure size 640x480 with 0 Axes>
```

The training stopped after the 37th epoch with no improvement.

Evaluation

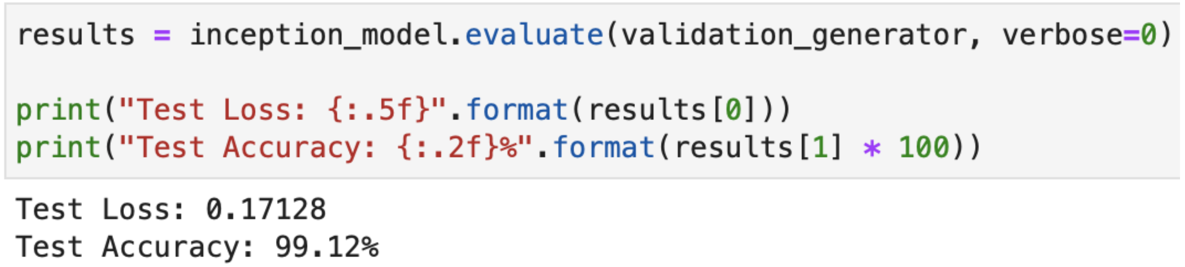
CNN



```
results = cnn_model.evaluate(validation_generator, verbose=0)

print("Test Loss: {:.5f}".format(results[0]))
print("Test Accuracy: {:.2f}%".format(results[1] * 100))
```

```
Test Loss: 0.34824
Test Accuracy: 89.88%
```

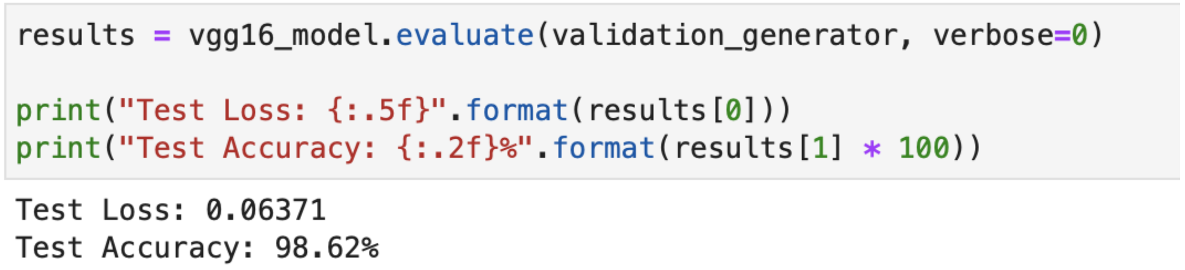
INCEPTION



```
results = inception_model.evaluate(validation_generator, verbose=0)

print("Test Loss: {:.5f}".format(results[0]))
print("Test Accuracy: {:.2f}%".format(results[1] * 100))
```

```
Test Loss: 0.17128
Test Accuracy: 99.12%
```

VGG16



```
results = vgg16_model.evaluate(validation_generator, verbose=0)

print("Test Loss: {:.5f}".format(results[0]))
print("Test Accuracy: {:.2f}%".format(results[1] * 100))
```

```
Test Loss: 0.06371
Test Accuracy: 98.62%
```

The InceptionV3 model performed better with a test accuracy of 99%

Deep Learning Analysis and Findings.

Deep Learning Analysis and Findings

After conducting experiments with three different models, we observed that the Inception-based model consistently outperformed the other two. Specifically, VGG16 exhibited a lightweight nature and faster training compared to the InceptionV3 model. Interestingly, InceptionV3 demonstrated quicker convergence than VGG16 during training. However, our plain custom model failed to converge effectively. In light of these findings, our final recommendation is to utilize the InceptionV3 Model due to its superior performance and relatively efficient convergence.

Model Flaws and Strength and Findings.

Model Flaws and Strength and findings

In our training process, we employed a limited number of training documents and relied on image augmentation techniques. This approach raises the concern of potential overfitting, where the model may become too specialized to the training data. To address this issue effectively, increasing the sample size for training data could be a beneficial step, providing a more diverse and representative dataset.

Advanced Steps

Additionally, aside from considering the Inception model, we might explore alternative architectures such as YOLO or ResNet to enhance model convergence. These models could offer improved performance and convergence characteristics, potentially further enhancing our model's ability to generalize to new data.

Thank you

WISDOM IZUCHUKWU ADIKE